Merchant & Gould

An Intellectual Property Law Firm

Merchant & Gould P.C.
1950 Independence Plaza
1050 Seventeenth Street
Denver, Colorado 80265-0100

TEL 303.357.1652
FAX 303.357.1671
www.merchantgould.com

Fax Transmission | March 30, 2007

To:	USPTO	From:	Jenifer Weck
Company:		Our Ref.:	40062.0264US01
Your Ref:	10/269072	Fax No.:	303.357.1671
Fax No.:	571.273.8300	Phone No.:	303.357.1652
Phone No.:		Total Pages:	1
State/Country:	USA	E-Mail:	jweck@merchantgould.com
Confirmation Via Mail:	□ Yes ☒ No	Return Fax To:	Jen Weck

Document Transmitted:

Message: Due to problems with the EFS Web accounting/payment regarding the issuance of an electronic filing receipt indicating the receipt of fees in the above matter, please charge Deposit Account No. 13-2725 the amount of $790 as fee for a Request for Continued Examination, and the amount of $120 as fee for a one-month extension of time ($910 total).

It is my understanding that once the EFS Web payment problems have been resolved Deposit Account 13-2725 will not be charged again for the payment of these fees.

This transmission contains information that is confidential and/or legally privileged. It is intended for use only by the person to whom it is directed. If you have received this telecopy in error, please notify us by telephone immediately so that we can arrange for the return of the original documents to us.

If you did NOT receive all of the pages, please call us in the U.S.A. at 303.357.1670 or fax us at 303.357.1671.

TABLE OF CONTENTS

1. REAL PARTY IN INTEREST

The real party in interest is Trend Micro, Inc.

2. RELATED APPEALS AND INTERFERENCES

There are no related appeals or interferences.

3. STATUS OF CLAIMS

These claims have been rejected and are appealed: claims 1-7, 9 and 11-30.
Claims 8 and 10 have been cancelled.
The claims on appeal are reproduced below in the Claims Appendix.

4. STATUS OF AMENDMENTS

No amendments were filed subsequent to final rejection.

5. SUMMARY OF CLAIMED SUBJECT MATTER

Recent (as of the priority date of the application) *second-generation* computer viruses are being written in an interpreted computer language (such as a scripting language) and are being distributed in source code (page 1, line 29-page 2, line 7; page 3, lines 1-6). By contrast, *first-generation* computer viruses were generally written in assembly language and distributed in binary code (page 2, lines 25-31; Figure 1). Prior art technology for detecting first-generation computer viruses is based on byte code matching algorithms where matches of suspect binary code with a virus pattern file indicates the presence of a virus. This prior technique can detect exact matches of the virus code, but is ineffective against a *polymorph* of a second-generation scripting virus (page 3, lines 17-22).

The reason that such prior techniques are ineffective is that a polymorph of a scripting virus includes small changes to the original virus code that changes the appearance of the code (*i.e.*, lexical or grammatical changes) but not its evil function. Thus, a byte-by-byte comparison would not match a virus pattern file, when in fact the virus is still present.

As such, the present invention is directed toward two embodiments: *generation* of a virus signature for a scripting virus and its polymorphs that can reliably detect the virus and its polymorphs (claims 5 and 15), and *detection* of a scripting virus and its polymorphs (claims 1 and 11).

5.1. Independent Claim 1

Claim 1 is a method for *identifying* a computer virus in interpreted language source code. The method identifies a computer virus by comparing a language-*independent* representation of the source code with a known virus signature. Use of a language-independent representation is beneficial because it can better detect polymorphs of scripting language viruses. If a language-*dependent* representation were used, the source code containing a virus would often not match the known virus signature because minor, cosmetic changes to the source code would be different than the virus signature. The second step of the claim generates the language-independent representation, the third step performs the comparison, and the fourth step determines if there is a match with a known virus signature, thus indicating that the source code has a computer virus.

Figure 3 illustrates source code 204, the generation of a language-independent representation using blocks 206, 204', 208 and 210. The language-independent representation is compared in a pattern matcher 218 to determine if a virus type 220 has been identified. Steps 302-314 of Figure 4 summarize this technique. Figure 9A is a polymorph of the source code shown in Figure 5A; minor changes have been made to the identifiers but the execution is still the same. For example, the function "F2" has been replaced by the word "Func2." Figure 11 shows the same source code that has now been processed into a language-independent form. The original source code has been tokenized as shown in Figure 9B; any language-dependent tokens are converted into language-independent tokens (page 11, lines 7-16). Figure 8 shows a dictionary of language-independent key actions 406.

5.2. Independent Claim 11

Claim 11 is a method for *identifying* a computer virus in interpreted language source code. The method identifies a computer virus by comparing a linearized, executing thread representation of the source code with a known virus signature. Use of a linearized, executing thread is beneficial; it can better detect polymorphs of scripting language viruses because cosmetic lexical and grammatical changes of a polymorph are eliminated. If the original source code itself were used, a match with a known virus signature might not occur because minor, cosmetic changes to the source code would be different than the virus signature.

Figure 9A illustrates interpreted language source code that is a polymorph of Figure 5A and that is potentially a computer virus. The second step of the claim requires parsing the source code into tokens; this result is shown in Figure 9B. The third step requires extracting selected key actions from the tokenized source code and an example is shown in Figure 10. In the fourth step the key actions are linearized to form an executing thread and an example is shown in Figure 11. The fifth step compares the executing thread with a known virus signature using a pattern matcher, for example, pattern matcher 218 and step 312. A match indicates a virus (virus type 220, step 314).

5.3. Independent Claim 5

Claim 5 is a method for *generating* a virus signature. Generally, blocks 204-212 illustrate the process for generating the virus signature. Steps 302-310 of Figure 4 summarize the technique. The second step of the claim requires parsing the source code into tokens using a parser; Figure 5B illustrates the source code (from Figure 5A) that has been parsed into tokens. The third and fourth steps require inputting the tokens into a threadizor and generating a linearized string of key actions that are language independent. Figure 6 illustrates using a key action dictionary to retrieve the key actions and Figure 7 illustrates linearized key actions for the source code example of Figure 5A. Thus, this linearized string of key actions may then be stored in a virus pattern file 214, shown in detail in Figure 8.

The advantage is that the identified computer virus is now represented in a virus pattern file as a linearized string of key actions that are language independent;

such a virus pattern is better able to detect polymorphs of the original virus because lexical and grammatical changes to the original virus are ignored and do not cause a false negative.

5.4. Independent Claim 15

Claim 15 is a method for *generating* a virus signature from interpreted language source code that includes a computer virus. Generally, blocks 204-212 illustrate the process for generating the virus signature. Steps 302-310 of Figure 4 summarize the technique. The second step of the claim requires parsing the source code into tokens using a parser; Figure 5B illustrates the source code (from Figure 5A) that has been parsed into tokens. The third step requires extracting key actions; Figure 6 illustrates an example of extracting key actions from the source code. An example of a dictionary of key actions is also shown in Figure 8 at 406 and key actions are described in the specification at page 10, lines of 7-11 and at page 11, lines 7-16.

The fourth step requires linearizing the key actions; Figure 7 illustrates linearized key actions for the source code example of Figure 5A. Once a set of minimum key actions are determined (fifth step), then the set of minimum key actions may be stored in a virus pattern file 214, shown in detail in Figure 8.

The advantage is that the identified computer virus is now represented in a virus pattern file as a linearized set of key actions; such a virus pattern is better able to detect polymorphs of the original virus because lexical and grammatical changes to the original virus are ignored and do not cause a false negative.

6. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

Ground I:

Claims 1-4, 11-14 and 21-24 have been rejected under 35 USC §102(e) as being anticipated by *Chandnani et al.*, U.S. Publication 2002/0073330 (*Chandnani*).

Ground II:

Claims 5-7, 9, 15-20 and 25-30 have been rejected under 35 USC §102(b) as being anticipated by *Chess et al.*, U.S. Patent No. 5,485,575 (*Chess*).

7. ARGUMENT

7.1. Ground I

7.1.1. *Claims 1 and 2*

Chandnani discloses a technique for detection of polymorphic script language viruses using lexical analysis. Figure 2 is useful for an understanding of the disclosure. In order to create virus detection data useful for later identification of a computer virus, samples of script language virus code are stored in sample store 56. The detection processor 52 then processes the sample code to create virus detection data that is stored in code detection database 57. Virus detection data includes token patterns for matching and CRC signature checking data. (Paragraphs 48-55.)

A great deal of effort is also spent creating language description data that describes a target script language. Script language processor 51 processes script language rules from rule base 54 to create script language description data that is stored in database 55. (Paragraphs 34-47.) In order to detect a computer virus in script language code a data stream representing that code is fed into detection engine 53 that uses the language description data 55 and the virus detection data 57 to determine if a virus is present. (Paragraphs 57-67.)

Important to an understanding of the distinction between *Chandnani* and the claimed invention is the nature of the data stream (representing the script language under analysis) that is fed into detection engine 53. Lexical analysis is used to convert the data stream into a stream of tokens, but there is no other processing performed upon the tokens and no further representation of the tokens is produced. (Paragraphs 60-62.) For example, paragraph 60 explains that the virus detection process has only two stages: tokenizing the data stream and processing the tokens using the detection data. Therefore, the tokens are, or represent, particular constructs in the script language being input and are therefore dependent upon that particular script language. In other words, because the first stage tokenizes the data stream and no other processing is performed, the tokens reflect that particular script language. As is known in the art of parsing, a token is an identifiable element in a script language.

Claim 1 is a method for identifying a computer virus in interpreted language source code. After a portion of the code is received, the second step of the claim requires "generating a language independent-representation of the portion of the interpreted language source code." The third and fourth steps also require the "language-independent representation" limitation.

By contrast, *Chandnani* does not teach or suggest generating a language-independent representation of the interpreted language source code to be scanned for a virus. *Chandnani* does show in Figure 2 that an incoming data stream representing script language is fed into a detection engine 53 that has a lexical analyzer. (Paragraphs 57-62.) The lexical analyzer converts the data stream into a stream of tokens but no other processing is performed on the tokens. A stream of tokens thus formed is not a language-*independent* representation of the source code. A stream of tokens is very much a language-*dependent* stream of characters, language constructs, symbols, *etc.*, that represent a particular scripting language.

The final Office action mailed August 23, 2006 in the "Response to Arguments" cites paragraphs 16, 20, 29, 51-56 and 64 of *Chandnani*. (Incidentally, this response to arguments concerning *Chandnani* is the exact same text presented in the "Response to Arguments" of the Office action mailed February 28, 2006.)

Paragraph 16 only discloses detecting a script language virus using language description data that is dependent on the script language. By contrast, claim 1 requires a language-independent representation. Paragraph 20 only discloses that the incoming data stream (the script language) is converted into a stream of tokens; paragraph 29 discloses lexical analysis of a data stream. Paragraphs 51-56 (and paragraph 50) describe creating viral code detection data (a detection regimen) from samples of script language viral code. The detection regimen is converted to binary format and then matched against suspected viral code (such as a corrupt system macro, as shown). But, this format is language dependent.

Because the second, third and fourth steps of claim 1 are not taught or suggested, it is requested that the rejection be withdrawn.

7.1.2. Claim 3

Claim 3 requires that "the virus signature is a language independent representation of an interpreted language source code computer virus." The data stream of tokens in *Chandnani* is compared to the virus detection data from database 57. But the virus detection data is not a language-*independent* representation of source code. The virus detection data is a pattern of tokens or a CRC signature check (Paragraph 51), neither of which is a language independent representation. Further, because the virus detection data is compared directly to the stream of tokens (the tokens representing the particular script language used), the virus detection data must also be in the form of the particular script language being used (Paragraph 64). If the virus detection data were in a language-*independent* form it would not be possible to match the stream of tokens that are in a language-*dependent* form. The Office cites paragraph 55 as disclosing that the virus signature is in a language-independent representation, but paragraph 55 only discloses that DFA data is used to transition from one stage in a pattern match operation to the next. The actual pattern match data (*i.e.*, the virus signature) is still in a language dependent form.

Because this limitation is not taught or suggested, it is requested that the rejection be withdrawn.

7.1.3. Claim 4

Claim 4 requires that both the source code and the virus signature "are presented as a linearized string of key actions." *Chandnani* does not disclose key actions nor a linearized string of such actions. In claim 4, the virus is identified by key actions in the source code, rather than the source code itself (page 9, lines 29-31 of the present application). The language-independent form of the source code is a linearized string of key actions, rather than the source code itself.

Because this limitation is not taught or suggested, it is requested that the rejection be withdrawn.

7.1.4. Claims 11-14, 21-24

Claim 11 requires "extracting selected key actions from the tokenized source code," "linearizing the key actions to generate an executing thread," and "comparing the executing thread with a virus signature of a known virus." As pointed out above with respect to *Chandnani*, once the incoming data stream is converted into a stream of tokens there is no other processing performed upon the tokens and no further representation of the tokens is produced. Paragraphs 57-64 of *Chandnani* only disclose that the incoming data stream is converted into a stream of tokens by a lexical analyzer; no further conversion or processing is performed. Thus, the required steps of "extracting selected key actions," "linearizing the key actions," and "comparing the executing thread" are not taught or suggested by the *Chandnani*. The Office relies upon paragraph 20, but paragraph 20 only discloses that the incoming data stream (the script language) is converted into a stream of tokens. The Office Action also cites paragraph 64, but this paragraph only discloses that a pattern is matched against the token stream; there is no "executing thread" because *Chandnani* does not linearize any key actions let alone the generate key actions from the tokens in the first place.

The final Office action mailed August 23, 2006 in the "Response to Arguments" cites paragraphs 16, 20, 29, 51-56 and 64 of *Chandnani.*

Paragraph 16 only discloses detecting a script language virus using language description data dependent on the script language. Paragraph 29 discloses lexical analysis of a data stream and nothing more. Paragraphs 51-56 (and paragraph 50) describe creating viral code detection data (a detection regimen) from samples of script language viral code. The detection regimen is converted to binary format and then matched against suspected viral code (such as a corrupt system macro, as shown). But, this format is language dependent.

Because these third fourth and fifth steps are not taught or suggested, it is requested that the rejection be withdrawn.

7.2. Ground II

7.2.1. *Claims 5-7 and 9*

Chess is not relevant to the claimed invention because *Chess* only discusses first-generation computer viruses that are written in assembly language and distributed in binary code. For example, Figures 4 and 5 of *Chess* show only that the virus is written in byte codes and throughout the specification reference is made to "blocks" or "bytes" (for example, column 2, line 40; column 6, lines 42-45). Nowhere in the specification of *Chess* is there any mention of an interpreted language, a scripting language, or distribution of a computer virus in source code.

Further, *Chess* describes a technique for identifying a virus and its characteristics by comparing a virus-infected computer program with the original, uninfected program. Claim 5 does not involve comparing a virus-infected program with the original uninfected program. Because *Chess* is not relevant to the claimed invention, it is not surprising that many of the features required by claim 5 are not taught or suggested by *Chess*.

Claim 5 requires "receiving a portion of interpreted language source code containing a computer virus." *Chess* does not teach or suggest an interpreted language that contains a computer virus nor source code that contains a virus. That portion of *Chess* relied upon only discloses host and infected files written in binary code or bytes. There is no discussion of source code or of an interpreted language.

Claim 5 also requires "parsing the portion of interpreted language source code into tokens using a parser." The computer virus code shown in Figure 4 of *Chess* is not interpreted language source code that is parsed into tokens.

Claim 5 also requires "generating a language-independent representation of the computer virus from said portion of the tokens." That portion of *Chess* relied upon does not disclose generating a representation of the computer virus that is independent of the language in which is written. In fact, Figure 4 simply shows manipulation of byte codes, the language in which viruses Sample 1 and Sample 2 are written. A byte code is not independent of the language used.

Claim 5 also requires that "the language independent representation is a linearized string of key actions." The Office Action relies upon column 12 of *Chess* for this feature. But this portion of *Chess* is referring to Figure 4 that simply shows that a computer virus is separated into sections of byte codes, some being variable over samples, some being constant. A byte code is not a "key action" as described in the specification of the instant application; nor is the string "linearized." As described in the specification of the instant application, "linearized" refers to rearranging portions of the virus code such that they appear in the order in which they are executed; Figure 4 simply shows that the byte codes of the computer virus are left as is.

Claim 5 also requires "storing the language-independent representation of the computer virus as a virus signature." Because *Chess* does not disclose a language-independent representation, it likewise does not disclose storing that representation as a virus signature. In fact, *Chess* does not concern itself with generating and storing a virus signature for future use. Column 11, lines 55-57 mentions that a signature can be extracted for identification, but there is no teaching that this signature is stored for later use.

The final Office action mailed August 23, 2006 in the "Response to Arguments" cites column 4, line 38 to column 5, line 2 and column 11, lines 39 to 56 of *Chess*. (Incidentally, this response to arguments concerning *Chess* is the exact same text presented in the "Response to Arguments" of the Office action mailed February 28, 2006.) But, columns 4 and 5 of *Chess* only disclose a comparison of a host computer file and an infected version of the host computer file. A virus-attachment description and a virus-structure description are prepared, but these required features of claim 5 are not disclosed.

Because these above steps and limitations are not taught or suggested, it is requested that the rejection be withdrawn.

7.2.2. Claims 15-20 and 25-30

Similar to claim 5, claim 15 also requires first and second steps of "receiving a portion of interpreted language source code" and "parsing the portion of interpreted language source code into tokens." Thesc two steps are not taught or suggested by *Chess* for the same reasons given above with respect to claim 5.

Further, claim 15 requires "extracting key actions," "linearizing key actions," "determining the set of minimum key actions," and "storing the set of minimum key actions as a virus signature." Respectfully, it is pointed out that columns 11 and 12 of *Chess* do not involve extracting, linearizing or determining key actions. Again, it is pointed out that Figure 4 of *Chess* simply discloses placing the virus code into sections and determining which sections remain constant over samples and which sections are variable. None of these four steps of claim 15 are taught or suggested in *Chess*.

The final Office action mailed August 23, 2006 in the "Response to Arguments" cites column 4, line 38 to column 5, line 2 and column 11, lines 39 to 56 of *Chess* as disclosing claims 5, 7-10, 15-20 and 25-30. But, columns 4 and 5 only disclose a comparison of a host computer file and an infected version of the host computer file. A virus-attachment description and a virus-structure description are prepared, but these required features of claim 15 are not disclosed.

Because these above steps and limitations are not taught or suggested, it is requested that the rejection be withdrawn.

8. CONCLUSION

In view of the foregoing, it is respectfully submitted that the above rejections are erroneous and it is requested that these rejections be reversed.

Respectfully submitted,
BEYER WEAVER LLP

Jonathan O. Scott
Registration No. 39,364

BEYER WEAVER LLP
Attorneys for Appellant

9. CLAIMS APPENDIX LISTING CLAIMS ON APPEAL

1. A method for identifying a computer virus in intcrpreted languagc source code using a virus scan engine, the method comprising:

receiving a portion of interpreted language source code;

generating a language-independent representation of the portion of the interpreted language source code;

comparing the language-independent representation with a virus signature in a pattern matcher; and

determining if the languagc-independent represcntation matches the virus signature, whereby a match indicates a computer virus has been identified.

2. The method of claim 1, wherein the interpreted language source code is a scripting language source codc.

3. The method of claim 1, wherein the virus signature is a language-independent representation of an interpreted language source code computer virus.

4. The method of claim 1, wherein the portion of interpreted language source code and the virus signaturc are represented as a linearized string of key actions.

5. A method for generating a virus signature using a virus scan engine, the method comprising:

receiving a portion of interpreted language source code containing a computer virus;

parsing the portion of interpreted language source code into tokens using a parser;

inputting at least a portion of the tokens into a threadizor;

generating a language-independent representation of the computer virus from said portion of the tokens using said threadizor, wherein said language independent representation is a linearized string of key actions; and

storing the language-independent representation of the computer virus as a virus signature.

generating the language-independent representation of the computer virus using

6. The method of claim 5, wherein the interpreted language source code is a scripting language source code.

7. The method of claim 5, wherein the virus signature is compiled in binary format.

8. (cancelled)

9. The method of claim 5, wherein the virus signature includes input from a virus analyst.

10. (cancelled)

11. A method for identifying a virus in interpreted language source code using a virus scan engine, the method comprising:

receiving a portion of interpreted language source code;

parsing the portion of the interpreted language source code into tokens to generate a tokenized source code using a parser, wherein at least some of the tokens represent key actions;

extracting selected key actions from the tokenized source code,

linearizing the key actions to generate an executing thread;

comparing the executing thread with a virus signature of a known virus in a pattern matcher; and

determining whether the executing thread matches the virus signature.

12. The method of claim 11, further comprising:

outputting the identification of the known virus.

13. The method of claim 11, wherein the portion of the interpreted language source code is lexically parsed.

14. The method of claim 11, wherein the portion of the interpreted language source code is lexically and grammatically parsed.

15. A method using a virus scan engine for generating a virus signature from a portion of interpreted language source code including a computer virus, the method comprising:

receiving a portion of interpreted language source code containing a computer virus;

parsing the portion of the interpreted language source code containing the computer virus into tokens to generate tokenized source code using a parser, wherein at least some of the tokens represent key actions;

extracting key actions from the tokenized source code,

linearizing the key actions to generate an executing thread using a threadizor;

determining the set of minimum key actions in the executing thread required to effect the computer virus; and

storing the set of minimum key actions as a virus signature.

16. The method of claim 15, further comprising:

compiling the virus signature in binary format.

17. The method of claim 15, further comprising:

compiling the virus signature with data input by a virus analyst; and

storing the virus signature as part of a virus pattern file.

18. The method of claim 17, wherein the virus pattern file further includes a dictionary of key actions.

19. The method of claim 15, wherein the portion of the interpreted language source code is lexically parsed.

20. The method of claim 15, wherein the portion of the interpreted language source code is lexically and grammatically parsed.

21. A computer readable medium containing program code for identifying a computer virus in interpreted language source code using a virus scan engine, the computer readable medium comprising instructions for:

receiving a portion of interpreted language source code;

parsing the portion of the interpreted language source code into tokens to generate a tokenized source code using a parser, wherein at least some of the tokens represent key actions;

linearizing at least a portion of the key actions to generate an executing thread;

comparing the executing thread with a virus signature of a known computer virus in a pattern matcher; and

determining whether the executing thread matches the virus signature.

22. The computer readable medium of claim 21, further comprising:

outputting the identification of the known computer virus.

23. The computer readable medium of claim 21, wherein the portion of the interpreted language source code is lexically parsed.

24. The computer readable medium of claim 21, wherein the portion of the interpreted language source code is lexically and grammatically parsed.

25. A computer readable medium containing program code using a virus scan engine for generating a virus signature from a portion of interpreted language source code including a computer virus, the computer readable medium comprising instructions for:

receiving a portion of interpreted language source code containing a computer virus;

parsing the portion of the interpreted language source code containing the computer virus into tokens to generate tokenized source code using a parser, wherein at least some of the tokens represent key actions;

linearizing at least a portion of the key actions to generate an executing thread using a threadizor;

determining the set of minimum key actions in the executing thread required to effect the computer virus; and

storing the set of minimum key actions as a virus signature.

26. The computer readable medium of claim 25, further comprising:

compiling the virus signature in binary format.

27. The computer readable medium of claim 25, further comprising:

compiling the virus signature with data input by a virus analyst; and

storing the virus signature as part of a virus pattern file.

28. The computer readable medium of claim 27, wherein the virus pattern file further includes a dictionary of key actions.

29. The computer readable medium of claim 25, wherein the portion of the interpreted language source code is lexically parsed.

30. The computer readable medium of claim 25, wherein the portion of the interpreted language source code is lexically and grammatically parsed.

10. EVIDENCE APPENDIX

No evidence has been submitted pursuant to §§ 1.130, 1.131, or 1.132 of 37 CFR, nor has any other evidence been entered by the examiner.

11. RELATED PROCEEDINGS APPENDIX

There have been no decisions rendered by a court or the Board in any related proceeding.